Living in the Compute Cloud

Jonathan Weiss
Peritor GmbH, www.peritor.com
Web 2.0 Expo Berlin, 2008
Peritor

• Consultancy from Berlin, Germany

• Specialized on Architecture, Scaling, and Ruby on Rails

• We help our clients adopt Cloud Computing
Motivation
Starting Point

One machine:

• Apache
• Ruby on Rails / PHP / Perl / Java / …
• MySQL
Worst Case Popularity

Problems

User
A Difficult Path
Problem: Backup

- High availability
- Redundancy
- Very big data sets
Problem: File System

- Important files have to be accessed by many servers
- NFS / Samba not practical
Problem: Database

• Scaling the database is really hard
• Replication setups are error-prone, complicated, and slow
Problem: Spontaneous Traffic
Problem: Load Fluctuation

![Graph showing load fluctuation from Monday to Sunday]
Don’t reinvent the wheel!
Cloud Computing Stacks
Cloud Computing Stacks

Amazon Web Services
- EC2
- S3
- SimpleDB

Google App Engine
- Application hosting
- BigTable
Amazon Web Services
Amazon Web Services

Simple Storage Service - S3
- Redundant Data Store
- $0,15 per GB data per month
- $0,10 - $0,20 per GB traffic

Elastic Compute Cloud - EC2
- Virtual server per hour
  - $0,10 - $0,80 per CPU hour
  - $0,10 - $0,20 per GB traffic

SimpleDB
- Hash-like database
- $0,14 per SDB machine hour
- $0,10 - $0,17 per GB traffic

E-Commerce..
S3 - Simple Storage Service

- Redundant storage - as much as you like
- max. 5 GB per object
- Organized in „Buckets“
- Web Service API for uploads
- Downloads via
  - Web Service
  - HTTP / HTTPS
  - BitTorrent
S3 - Buckets

- Unique over all S3
- Contains many key-value-metadata tuples
- Cannot contain other buckets!
- Key can contain ‘/’
EC2 - Elastic Compute Cloud

Based on XEN virtualization

On demand virtual servers - controlled with Web Service API

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>L</th>
<th>XL</th>
<th>HighCPU M</th>
<th>HighCPU XL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU / EC2 Compute Units</td>
<td>1 x 1</td>
<td>2 x 2</td>
<td>4 x 2</td>
<td>2 x 2.5</td>
<td>8 x 2.5</td>
</tr>
<tr>
<td>RAM</td>
<td>1.7 GB</td>
<td>7.5 GB</td>
<td>15 GB</td>
<td>1.7 GB</td>
<td>7 GB</td>
</tr>
<tr>
<td>HDD</td>
<td>160 GB</td>
<td>850 GB</td>
<td>1690 GB</td>
<td>350 GB</td>
<td>1690 GB</td>
</tr>
<tr>
<td>Platform</td>
<td>32 bit</td>
<td>64 bit</td>
<td>64 bit</td>
<td>32 bit</td>
<td>64 bit</td>
</tr>
</tbody>
</table>

Use your favorite Linux distro (Linux 2.6.16)
**EC2 - Extras**

- ACLs for hosts/ports access control
- Choose your favorite datacenter (currently only US)
- Elastic IP for dynamic reconfiguration of your endpoints
- Elastic Block Store
  - Persistent storage
  - Mountable as a filesystem (by one host at a time)
  - Snapshots to S3
EC2 Tools

List available images

> ec2-describe-images

Start a new instance

> ec2-run-instances ami-5bae4b32 -k $RSA_KEY

Login with SSH

> ssh root@domU-12-34-31-00-00-05.usma1.compute.amazonaws.com

Shutdown instance

> ec2-terminate-instances i-10a64379
SimpleDB

- Hash-like database
- Stores items with attribute-value pairs
- Very different to the relational model of SQL databases
- Meant for small items, use S3 for binaries/files and store the URL in SimpleDB
- Organized into ‘domains’
- Redundant and distributed
SimpleDB Data

• No schema
• Everything is a string
  • Pad numbers with zeros 000098.5
  • Use ISO 8601 format for dates 2008-10-22T 11:10+02:00CET
• Allows list values – multiple values for one attribute
• No joins between domains
• SQL-like queries to retrieve data

['Make' = 'Audi'] intersection ['Model' = 'S4']
SimpleDB Example

Sample Domain ‘products’

<table>
<thead>
<tr>
<th>ItemID</th>
<th>Name</th>
<th>Description</th>
<th>Available Colors</th>
<th>Created At</th>
<th>Inventory</th>
</tr>
</thead>
<tbody>
<tr>
<td>721911</td>
<td>Blue-Shirt</td>
<td>A blue shirt</td>
<td>Blue</td>
<td>2008-10-01</td>
<td>0000312</td>
</tr>
<tr>
<td>127128</td>
<td>Dreamcar</td>
<td>A nice car</td>
<td>Blue, Red</td>
<td>2007-09-11</td>
<td>0047639</td>
</tr>
<tr>
<td>127181</td>
<td>PC</td>
<td>An old PC</td>
<td>Grey</td>
<td>2008-07-21</td>
<td>0000931</td>
</tr>
<tr>
<td>857314</td>
<td>Notebook</td>
<td>100 pages</td>
<td>Red, Yellow, Green</td>
<td>2007-12-17</td>
<td>0087631</td>
</tr>
</tbody>
</table>
Google App Engine
Google App Engine

- Run your application directly on Google’s infrastructure
- No concept of hardware – just deploy your application
- At the moment limited to Python
- Free for up to 500MB and CPU for ~ 5mio page impressions
Python Runtime

- Python 2.5.2
- SDK for local development
- Restricted on some low-level modules
Google APIs

Compensation for not having access to lower levels like sockets:

- Memcache
- Image
- Email
- URL fetch
- Google Auth & User
BigTable

Very similar to SimpleDB
  • No schema
  • List values

But also very different
  • Data type support
  • References and multiple tables
  • Blob fields (1MB)
BigTable in App Engine

Python bindings and GQL

```python
class Post(db.Model):
    title = db.StringProperty(required=True)
    text = db.TextProperty()

class Comment(db.Model):
    post = db.ReferenceProperty(Post)
    comment_text = db.TextProperty()
    user = db.UserProperty()

my_post = Post.gql('WHERE id = :1', title).get()```
App Engine Gotchas

• All requests are killed after a few seconds
• No background jobs
• No possibility to backup/snapshot data
• Emails can only be sent from Google accounts
• Restricted to pure-Python libraries and given APIs
How does this solve my problems?
S3 - Backup

• s3sync.rb
• Brackup
• Jungle Disk
• S3 FUSE
• s3DAV
• Duplicity

• S3Browser
• Firefox S3 Organizer extension
• ...

peritor consulting
**s3sync.rb**

**Backup**

```bash
export AWS_ACCESS_KEY_ID='AAAAAAAAAAAAAAAAA'
export AWS_SECRET_ACCESS_KEY='BBBBBBBBBBBBBBBBBBBBBBBBBBBB'
export SSL_CERT_FILE='/home/amazon/s3sync/ssl.crt'

./s3sync.rb -v -r --ssl /usr/local/backups MyBucket:/www.example.com/backups
```

**Restore**

```bash
./s3sync.rb -v -r --ssl MyBucket:/www.example.com/backups /usr/local/backups
```
S3 Asset Host

Client

static requests

S3

www.example.com

dynamic requests
S3 - Authenticated User Data

1) GET /file_uploads/12
2) Web Service requests
3) REDIRECT
4) GET /my_bucket/file_uploads/......

Client

www.example.com
On-Demand Computing with EC2

Time based, e.g. with cron

```bash
> crontab -l -u ec2
0 8 *** /home/ec/start_additional_instances.sh
0 18 *** /home/ec/stop_additional_instances.sh
```
On-Demand Computing with EC2

Load based, e.g. with Monit

```
> tail /etc/monitrc

check system example.com
    if cpu usage (user) > 70% for 5 cycles then
        exec "/home/ec2/add_additional_instance.sh"
    if cpu usage (user) < 50% for 10 cycles then
        exec "/home/ec2/remove_additional_instance.sh"
```
On-Demand Computing with EC2
On-Demand Computing with EC2
Using SimpleDB & S3
EC2 for extra capacity

Your Datacenter

apache load balancer

App Server

DB

VPN

Amazon Datacenter

App Server EC2

App Server EC2

App Server EC2
App Engine
Real Life Use Cases
GoodBad.me

- Twitter-Mashup for RailsRumble 2008
- Needed test/development servers and performance benchmarking infrastructure
- Conventional hosters too inflexible and slow
DaWanda.com

- Marketplace for handmade goods
- Large amount of uploaded user data
- Use of S3 to scale and cope with asset load
G.ho.st

- Virtual desktop/computer in the browser
- Running completely on EC2 and S3
Final Thoughts
Final Thoughts

• Get accustomed to “Eventual Consistency”
• Be prepared to leave relational databases
• Many miss strong SLAs – most of the time you can live fine without
• Hardware is a commodity – only specialize in it if really necessary